Apple Assembly Line

$1.80

Volume 6 -- Issue 8 May, 1986

In This Issue...



## Enhancing Applesoft with the Toolbox Series

A number of years ago, when Roger Wagner Publishing was still called Southwestern Data Systems, he published Peter Meyer's program "The Routine Machine". The system evolved into four packages: Wizard's Toolbox, Database Toolbox, Video Toolbox, and Chart'n Graph Toolbox. Each "Toolbox" contains a large assortment of assembly language routines which enhance the capabilities of Applesoft. The "Workbench" (included with each Toolbox) allows programmers to add any assortment of these routines to their Applesoft programs at any time. The routines are all called by using the ampersand (&) statement.

Roger will make a special deal for Apple Assembly Line subscribers: he'll send a free copy of the "Trial-size Toolbox" (normally $3) to anyone who mentions reading about the package here. The disk includes eight ampersand commands, including a charting command-set with 12 sub-commands, a fixed-length input command, and a print with word-wrap command. All are usable under either DOS 3.3 or ProDOS. Also on the disk is the text of a 50-page manual. The manual includes a tutorial for the toolbox system, a complete explanation of the commands included on the sampler disk, and a comprehensive listing of every command in each of our Toolbox packages. For the free sampler write to Roger Wagner Publishing, Box 582, Santee, CA 92071.

The Toolbox packages are normally $39.95 each. We'll sell them here at S-C for $36 each, or $140 for the complete set.

DOS 3.3 for the UniDisk 3.5 (RWTS 3.5).............Bill Morgan

We finally got one of Apple's new UniDisk 3.5 drives for the //e, and let me tell you it's very nice. This small but large addition to our favorite computer is about half the volume of a Disk II, but each disk stores almost six times as much information. It's even a bit faster than the 5.25" drives, about 1.3 times the speed.

Of course there's a catch. In line with Apple's policy of supporting ProDOS only, the new device doesn't use DOS 3.3, at least not as far as Apple is concerned. There are already several different UniDisk versions of DOS, and we're about to build our own right here. It's really quite easy.

There are two parts to the problem: intercepting and handling RWTS calls to the UniDisk slot, and formatting a 3.5" disk with a DOS VTOC and Catalog.

There are a variety of ways to take over a call to RWTS. When we call RWTS at $3D9 it jumps on to $B7B5, where interrupts are disabled before calling the real RWTS entry at $BD00. Some programs take control at $B7B7 and others at $BD00. I looked at the code at $BD00 and saw that it does a little housekeeping and then at $BD10 loads the accumulator with the slot*16 value from the IOB. That looks like the ideal time to check to see if this call is for my slot, so $BD12 is where I patch in the jump to my code. If you are using several nonstandard devices with DOS 3.3 (Sider or other hard disk, RAM disk, other drives) you will need to keep track of who's patching into RWTS where.

Now we come to the question of where to put our version of RWTS. There's certainly no room inside DOS for almost a page of code plus two pages of buffer. I thought I could probably squeeze the code into page three, but that still left that buffer (not to mention the crowd already living at that popular address!) It occurred to me to throw INIT away and put the code inside the existing RWTS at $BEAF, but what about the buffer? I finally decided to use the time-honored technique of moving the DOS buffers and HIMEM down and installing my program and buffer in there. That's also crowded, but where isn't? The first working version of RWTS 3.5 ran at $9900, with the buffer at $9B00-9CFF. The installation routine checked to see if anyone else was using the space and returned an error if so. Applesoft and the S-C Macro Assembler got along with this arrangement just fine, so I spent some time polishing the program and started to write this article.

That's when I was forcibly reminded that the S-C Word Processor sets its own HIMEM and is firmly convinced that $9900-99FF is the buffer for characters deleted off the screen. In other words, the first time I tried to save some text to the UniDisk it blew sky high. I had decided to live without the Word Processor on the UniDisk for the time being when I noticed a couple of interesting things in Beneath Apple DOS. There is a 342-byte buffer inside RWTS at $BB00-BC55, and the code immediately after that buffer is called only by INIT! There really are two full pages of available buffer space inside DOS

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners....................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM..................without source code $30, with source $50
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
"Bag of Tricks 2"...................................($49.95) $45 *
Applesoft Toolbox Series, Roger Wagner Publishing......each ($39.95) $36 *
all four ($159.80) $140 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
Each disk contains the source code from three issues of AAL,
saving you lots of typing and testing.
The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler. Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 20 for $20 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100 *
Envelopes for Diskette Mailers............................ 6 cents each

65802 Microprocessor (Western Design Center)..................($95) $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG)..........................................($149.50) $140 *
Switch-a-Slot (SCRG).......................................($179.50) $170 *

"65816/65802 Assembly Language Programming", Fischer........($19.95) $18 *
"Programming the 65816", Eyes...............................($22.95) $21 *
"Apple //e Reference Manual", Apple Computer................($24.95) $23 *
"Apple //c Reference Manual", Apple Computer................($24.95) $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95) $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95) $18 *
"Apple ProDOS: Advanced Features for Programmers", Little..($17.95) $17 *
"Inside the Apple //c", Little..............................($19.95) $18 *
"Inside the Apple //e", Little..............................($19.95) $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95) $18 *
"Apple ][ Circuit Description", Gayler.......................($22.95) $21 *
"Understanding the Apple II", Sather........................($22.95) $21 *
"Understanding the Apple //e", Sather.......................($24.95) $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95) $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95) $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95) $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95) $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95) $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95) $9 *
"Microcomputer Graphics", Myers.............................($14.95) $14 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95) $18 *
"Assembly Lines -- the Book", Wagner........................($19.95) $18 *
"AppleVisions", Bishop & Grossberger........................($39.95) $36 *

* On these items add $2.00 for the first item and
$.75 for each additional item for US shipping.
Foreign customers inquire for postage needed.
Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** Master Card, VISA, Discover and American Express ***

along with room for the code.

So this edition of RWTS 3.5 runs at $BEAF, with its buffer at $BB00-BCFF. I did hit one more snag when I went to use that buffer area; $BCDF-BCFF is officially unused, which means it's a popular place for other patches. My system has part of our fast LOAD/BLOAD patch (AAL April 83) there, so I had to shave a few more bytes out of my program to make room to move the LOAD patch up to $BF97-BFB7. You may have to make some such adjustment, so be sure to check for some other patch at $BCDF.

The UniDisk 3.5 uses a new software interface, called the Protocol Converter. The PC is a sort of serial bus, which can have several devices daisy-chained to the same controller. We program the PC with a calling structure very similar to the ProDOS MLI calls. Here's an example:

```
CALL  JSR DISPATCH
      .DA #1          read command
      .DA PARMLIST
      BCS ERROR
      ... whatever code

PARMLIST
      .DA #3          3 parameters
      .DA #1          unit number
      .DA BUFFER      buffer address
      .DA <BLOCK      block number (3 bytes)
```

That's all it takes to read a 512-byte block into our buffer. Notice that this standard specifies a 3-byte block number: all current devices use only two bytes of the block number, but they're allowing for expansion beyond 32 megabytes. The unit number isn't the same as a ProDOS unit; this is the position of the device in the PC chain. We need to look up the value of DISPATCH in the card. The byte at $CsFF (s = slot) contains the offset into the ROM of the ProDOS driver entry and the Protocol Converter entry is defined to be 3 bytes after that. For example. in my UniDisk 3.5 controller in slot 5 the byte at $C5FF is $0A. That means that the ProDOS entry to the card is $C50A and the PC entry is $C50D.

There's a quick look at the Protocol Converter. We haven't seen much information published about it yet. The new //c Technical Reference Manual has a good section, including a ROM listing, but the //e UniDisk 3.5 includes no programmer's documentation. Bob is planning a more extensive article on its programming for next month's AAL. Stay tuned...

Apple's new memory expansion card has a PC interface and this RWTS will work with that card as well, but some modification will be needed to use more than one PC at a time. The installation code could scan all slots looking for PCs and build a table of valid slots and entry addresses. Then the initial code at MY.RWTS could search that table and plug the appropriate PC.DISPATCH address into the calls.

The Protocol Converter sees the UniDisk as 1600 blocks of 512

bytes each, for a total of 819,200 (800K) bytes of storage. We have no way to find out about actual tracks and sectors on the disk; this drive seems to use the Macintosh scheme of a variable number of blocks per track. Therefore, we're going to translate DOS's tracks and sectors into some block number and ask the PC for that block, not worrying about where it actually comes from.

The VTOC on a DOS disk has room for 50 tracks of 32 sectors each. That adds up to 400K, or exactly half a UniDisk, so we should be able to set things up with 2 logical drives of 400K each. The number of tracks per disk and the number of sectors per track are both stored as parameters in the VTOC as well, just to make things easier. Two drives per disk means that we can put drive one in the lower 800 blocks and drive two in the upper 800. Figuring that 32 sectors per track means 16 blocks per track and two sectors per block gives us this equation:

BLOCK = (DRIVE-1)*800 + TRACK*16 + SECTOR/2

An even-numbered sector is in the lower half of a block, odd in the upper half.

Since each sector is half of one block on the disk, we can't just write one sector. We have to read a block, copy the new information into half of the buffer, then write that block back out. This takes extra time, but simplifies some of the control logic because every call does a read first.

That first working version of RWTS 3.5 did a new read for every read call, and a new read and write for every write. Well that proved to be much too slow, even slower than the old Disk II. Then I realized that nearly all DOS operations are reading or writing consecutive sectors in a file, so I must be spending a lot of time reading a block that was already in my buffer just to get the sector in the other half of the block. Sure enough, the performance almost doubled when I started keeping track of which block was in the buffer and skipping re-reads of the same block. It does seem to be a good idea to make a special case of the VTOC sector and always re-read that one, just in case we change disks after writing the VTOC as the last operation on the old disk.

## Line by Line

In the INSTALL routine we first make sure there is a Protocol Converter in the slot this RWTS expects. If so, we patch in the JMP to our code near the beginning of the normal RWTS and disable INIT by patching an RTS instruction at the beginning of the command handler. MOVE then puts our routine into place at $BEAF and looks up the PC entry point into the ROMS and installs that address into the instructions that call the interface card. NO.PC provide an error message if we can't find a PC. The ID.TABLE has the bytes which mark a PC interface, interspersed with $FFs so we can use the same index for the ROM and the table.

The meat of the program begins at MY.RWTS. We enter here with slot*$10 in the A register so we can check to see if we need to handle this call. If not we execute the instructions we overwrote with the JMP and go back to the normal RWTS. If is is our call, the first thing we do at MINE is check to see if we handled the last RWTS call as well. If so, all is well, but if normal RWTS was used last then it clobbered the buffer at $BB00. We therefore trash LAST.BLOCK so the tests down at CHECK.FOR.RE.READ will be forced to read a new block.

SET.BLOCK tranforms the requested track and sector into a block number, in the process setting carry to indicate whether we want the high or low half of the block. SET.POINTERS then creates two pointers for MY.BUFFER and IOB.BUFFER, using that carry bit along the way. At SET.DRIVE we check which drive is called for and modify BLOCK to read the other half of the diskette if it says drive 2. While we're at it, we plug the drive number into the volume number found, so it will appear as the volume number in a CATALOG. SET.COMMAND gets the command and makes sure it's either READ or WRITE. Anything else becomes a NOP.

At CHECK.FOR.RE.READ we compare the block number requested with the number of the block in the buffer and if they're different we go on to read the new block. If we already have the block we need, CHECK.FOR.VTOC double-checks to see if it's a VTOC we're reading. If so, we need to re-read it anyway, in case it's now a different disk in the drive. Once all that rigamarole is out of the way, the eight bytes at READ are all it takes to actually read the block!

At SKIP.READ we get the command again. (I just noticed that we can move the SET.COMMAND code to this point, since doing an extra READ won't hurt anything, even if the command is bad. That way we can eliminate MY.COMMAND and its STA and LDA instructions. Furthermore, changing the CMP #2 to an LSR and changing the BEQ to a BCC shaves out another byte, for a total of five fewer bytes. There's always more space to be found!) If the command is a READ then READ.MOVE.BUFFER copy MY.BUFFER into the IOB's buffer and we're done. If it's a WRITE, WRITE.MOVE.BUFFER copies the other way, from the IOB buffer into mine, and then calls the ROM to write out the block. Then GOOD.EXIT clears carry and loads a return code of zero before branching to the end. ERROR.EXIT loads up either WRITE PROTECT or DRIVE ERROR and sets carry before returning to the caller.

## FORMAT 3.5 ---

Since we threw away INIT to fit all this inside of DOS, and since the standard INIT wouldn't put enough VTOC or CATALOG space on the disk, we're also going to need a special FORMAT program.

There are two stages in the process of formatting a disk: initializing all the tracks with address information; and writing the VTOC, empty catalog track, and boot program. Initializing a Protocol Converter device is easy, just call the

PC and let it do all the work. Then we can use our nice new RWTS to write all the rest of the necessary data. Just be sure that RWTS 3.5 is installed before calling FORMAT 3.5.

Since this catalog track is 31 sectors long there is room for 217 files instead of the normal 105. Other than the length, the structure is exactly the same as a normal DOS catalog. The differences in the VTOC are bytes $34-35, the number of tracks per disk and sectors per track, and the bitmap. The bitmap skips tracks $0 and $11, fills all four bytes per track rather than alternate pairs, and extends all the way to the end of the sector.

The boot program here is just a quick message. I hope to have a real boot loader ready for next month's AAL.

```
                    1000 *SAVE S.UNIDISK RWTS
                    1010 *--------------------------------
05-                 1020 UNIDISK.SLOT          .EQ 5
                    1030
26-                 1040 MY.COMMAND            .EQ $26
3C-                 1050 MY.BUFFER.POINTER     .EQ $3C
3E-                 1060 IOB.BUFFER.POINTER    .EQ $3E
48-                 1070 IOB.PTR               .EQ $48
                    1080
BB00-               1090 MY.BUFFER             .EQ $BB00
                    1100
BD12-               1110 PATCH.POINT           .EQ $BD12
BD15-               1120 PATCH.RETURN          .EQ $BD15
                    1130
C500-               1140 PC.DISPATCH           .EQ UNIDISK.SLOT*$100+$C000
                    1150
FDDA-               1160 PRBYTE                .EQ $FDDA
FDED-               1170 COUT                  .EQ $FDED
                    1180 *--------------------------------
                    1190        .OR $803
                    1200        .TF RWTS 3.5
                    1210
                    1220 INSTALL
0803- A2 06         1230        LDX #6           make sure we have a
0805- BD 61 08      1240 .1     LDA ID.TABLE,X   protocol converter
0808- DD 01 C5      1250        CMP UNIDISK.SLOT*$100+$C001,X
080B- D0 31         1260        BNE NO.PC
080D- CA            1270        DEX
080E- CA            1280        DEX
080F- 10 F4         1290        BPL .1
                    1300
0811- A9 4C         1310        LDA #$4C         patch in the JMP
0813- 8D 12 BD      1320        STA PATCH.POINT  to our code
0816- A9 AF         1330        LDA #MY.RWTS
0818- 8D 13 BD      1340        STA PATCH.POINT+1
081B- A9 BE         1350        LDA /MY.RWTS
081D- 8D 14 BD      1360        STA PATCH.POINT+2
0820- A9 60         1370        LDA #$60
0822- 8D 4F A5      1380        STA $A54F        disable INIT
                    1390
0825- A0 E8         1400 MOVE   LDY #IMAGE.SIZE+1 install our code
0827- B9 67 08      1410 .1     LDA IMAGE-1,Y
082A- 99 AE BE      1420        STA MY.RWTS-1,Y
082D- 88            1430        DEY
082E- D0 F7         1440        BNE .1
                    1450
0830- 18            1460        CLC
0831- AD FF C5      1470        LDA UNIDISK.SLOT*$100+$C0FF
0834- 69 03         1480        ADC #3           find protocol
0836- 8D 48 BF      1490        STA READ.CALL    converter entry
0839- 8D 6A BF      1500        STA WRITE.CALL
083C- D0 0D         1510        BNE DONE         ...always
                    1520
```

```
083E- A2 00    1530 NO.PC   LDX #0
0840- BD 4E 08 1540 .1      LDA MESSAGES,X    print an error message
0843- F0 06    1550         BEQ DONE
0845- 20 ED FD 1560         JSR COUT
0848- E8       1570         INX
0849- D0 F5    1580         BNE .1
084B- 4C D0 03 1590 DONE    JMP $3D0
               1600 *--------------------------------
               1610 MESSAGES
084E- 8D       1620         .HS 8D
084F- CE EF A0
0852- D0 C3 A0
0855- E9 EE A0
0858- F3 EC EF
085B- F4 A0    1630         .AS -/No PC in slot /
085D- B5       1640         .DA #$B0+UNIDISK.SLOT
085E- 87 8D 00 1650         .HS 878D00
               1660 *--------------------------------
0861- 20 FF 00
0864- FF 03 FF
0867- 00       1670 ID.TABLE .HS 20.FF.00.FF.03.FF.00
               1680 *
               1690 *          Protocol Converter ID Bytes
               1700 *--------------------------------
0868-          1710 IMAGE   .EQ *
               1720         .PH $BEAF
               1730 MY.RWTS
BEAF- C9 50    1740         CMP #UNIDISK.SLOT*$10
BEB1- F0 06    1750         BEQ MINE          my call!
BEB3- AA       1760         TAX               not mine, so do
BEB4- A0 0F    1770         LDY #$F           patched-over code
BEB6- 4C 15 BD 1780         JMP PATCH.RETURN  and go back
               1790 *--------------------------------
               1800 MINE
BEB9- A0 0F    1810         LDY #$F
BEBB- D1 48    1820         CMP (IOB.PTR),Y   check previous slot
BEBD- F0 07    1830         BEQ SET.BLOCK     same, so go on
BEBF- 91 48    1840         STA (IOB.PTR),Y   set previous slot
BEC1- A9 FF    1850         LDA #$FF
BEC3- 8D 8C BF 1860         STA LAST.BLOCK    trash LAST.BLOCK
               1870
               1880 SET.BLOCK
BEC6- A9 00    1890         LDA #0
BEC8- 8D 8A BF 1900         STA BLOCK+1
BECB- A0 04    1910         LDY #4
BECD- B1 48    1920         LDA (IOB.PTR),Y   get track
BECF- 0A       1930 .1      ASL
BED0- 2E 8A BF 1940         ROL BLOCK+1       *16
BED3- 88       1950         DEY
BED4- D0 F9    1960         BNE .1
BED6- 8D 89 BF 1970         STA BLOCK
BED9- A0 05    1980         LDY #5
BEDB- B1 48    1990         LDA (IOB.PTR),Y   get sector
BEDD- 4A       2000         LSR               /2, odd/even into carry
BEDE- 0D 89 BF 2010         ORA BLOCK
BEE1- 8D 89 BF 2020         STA BLOCK
               2030
               2040 SET.POINTERS
BEE4- A9 00    2050         LDA #MY.BUFFER
BEE6- 85 3C    2060         STA MY.BUFFER.POINTER
BEE8- A9 BB    2070         LDA /MY.BUFFER
BEEA- 69 00    2080         ADC #0       carry sets hi/lo half of buffer
BEEC- 85 3D    2090         STA MY.BUFFER.POINTER+1
BEEE- A0 08    2100         LDY #8
BEF0- B1 48    2110         LDA (IOB.PTR),Y   get IOB buffer
BEF2- 85 3E    2120         STA IOB.BUFFER.POINTER
BEF4- C8       2130         INY
BEF5- B1 48    2140         LDA (IOB.PTR),Y
BEF7- 85 3F    2150         STA IOB.BUFFER.POINTER+1
               2160
               2170 SET.DRIVE
BEF9- A0 02    2180         LDY #2
BEFB- B1 48    2190         LDA (IOB.PTR),Y   get drive
BEFD- A0 10    2200         LDY #$10
BEFF- 91 48    2210         STA (IOB.PTR),Y   set previous drive
BF01- 88       2220         DEY
BF02- 88       2230         DEY
BF03- 91 48    2240         STA (IOB.PTR),Y   set previous volume
```

```
BF05- 4A                  LSR
BF06- B0 10               BCS SET.COMMAND    .CS. if D1
BF08- AD 89 BF            LDA BLOCK          add 800 to BLOCK if D2
BF0B- 69 20               ADC #800
BF0D- 8D 89 BF            STA BLOCK
BF10- AD 8A BF            LDA BLOCK+1
BF13- 69 03               ADC /800
BF15- 8D 8A BF            STA BLOCK+1

                SET.COMMAND
BF18- A0 0C               LDY #$C
BF1A- B1 48               LDA (IOB.PTR),Y    get command
BF1C- F0 53               BEQ GOOD.EXIT
BF1E- C9 03               CMP #3             exit if not READ or WRITE
BF20- B0 4F               BCS GOOD.EXIT
BF22- 85 26               STA MY.COMMAND     save command

                CHECK.FOR.RE.READ
BF24- A2 00               LDX #0             zero the flag
BF26- A0 01               LDY #1             check two bytes
BF28- B9 89 BF  .1        LDA BLOCK,Y
BF2B- D9 8C BF            CMP LAST.BLOCK,Y   compare
BF2E- F0 04               BEQ .2             same, so go on
BF30- E8                  INX                different, so flag it
BF31- 99 8C BF            STA LAST.BLOCK,Y   and store new value
BF34- 88        .2        DEY
BF35- 10 F1               BPL .1             now do low bytes
BF37- 8A                  TXA                check the flag
BF38- D0 0D               BNE READ           if different, go read

                CHECK.FOR.VTOC
BF3A- A0 05               LDY #5
BF3C- B1 48               LDA (IOB.PTR),Y    get sector
BF3E- D0 0F               BNE SKIP.READ      non-zero isn't VTOC
BF40- 88                  DEY
BF41- B1 48               LDA (IOB.PTR),Y    get track
BF43- C9 11               CMP #$11
BF45- D0 08               BNE SKIP.READ      not $11 isn't VTOC

BF47- 20 00 C5  READ      JSR PC.DISPATCH
BF48-           READ.CALL .EQ *-2
BF4A- 01                  .DA #1             READ
BF4B- 85 BF               .DA PARMLIST
BF4D- B0 27               BCS ERROR.EXIT

                SKIP.READ
BF4F- A5 26               LDA MY.COMMAND     check command
BF51- C9 02               CMP #2
BF53- F0 0B               BEQ WRITE.MOVE.BUFFER

                READ.MOVE.BUFFER
BF55- A0 00               LDY #0
BF57- B1 3C     .1        LDA (MY.BUFFER.POINTER),Y
BF59- 91 3E               STA (IOB.BUFFER.POINTER),Y
BF5B- C8                  INY
BF5C- D0 F9               BNE .1
BF5E- F0 11               BEQ GOOD.EXIT      ...always

                WRITE.MOVE.BUFFER
BF60- A0 00               LDY #0
BF62- B1 3E     .1        LDA (IOB.BUFFER.POINTER),Y
BF64- 91 3C               STA (MY.BUFFER.POINTER),Y
BF66- C8                  INY
BF67- D0 F9               BNE .1

BF69- 20 00 C5  WRITE     JSR PC.DISPATCH
BF6A-           WRITE.CALL .EQ *-2
BF6C- 02                  .DA #2             WRITE
BF6D- 85 BF               .DA PARMLIST
BF6F- B0 05               BCS ERROR.EXIT

                GOOD.EXIT
BF71- 18                  CLC
BF72- A9 00               LDA #0
BF74- F0 0A               BEQ EXIT           ...always

```

```
                ERROR.EXIT
BF76- C9 2B             CMP #$2B     write protect?
BF78- F0 03             BEQ .1
BF7A- A9 40             LDA #$40     make everything else DRIVE ERROR
BF7C- 2C                .HS 2C
BF7D- A9 10     .1      LDA #$10
BF7F- 38                SEC

BF80- A0 0D     EXIT    LDY #$D
BF82- 91 48             STA (IOB.PTR),Y   save return code
BF84- 60                RTS
                *--------------------------------
                PARMLIST
BF85- 03                .DA #3         3 parameters
BF86- 01                .DA #1         unit number
BF87- 00 BB             .DA MY.BUFFER  buffer address
BF89-           BLOCK   .BS 3          block number

BF8C- FF FF     LAST.BLOCK .HS FFFF
                *--------------------------------
BF8E-                   .BS $BF97-*
                        .EP
094F-           IMAGE.END .EQ *-1
E7-             IMAGE.SIZE .EQ IMAGE.END-IMAGE
```

```
                *SAVE S.FORMAT.UNIDISK
                *--------------------------------
05-             UNIDISK.SLOT .EQ 5

03D9-           RWTS         .EQ $3D9

C500-           PC.DISPATCH  .EQ UNIDISK.SLOT*$100+$C000

FC58-           HOME         .EQ $FC58
FDED-           COUT         .EQ $FDED
                *--------------------------------
                        .OR $803
                        .TF FORMAT.UNIDISK

0803- 18        FORMAT  CLC
0804- AD FF C5          LDA UNIDISK.SLOT*$100+$C0FF
0807- 69 03             ADC #3
0809- 8D 0D 08          STA PC.CALL
080C- 20 00 C5          JSR PC.DISPATCH   format the disk
080D-           PC.CALL .EQ *-2
080F- 03                .DA #3
0810- 9E 08             .DA PC.PARMS
0812- B0 7F             BCS ERROR
0814- A9 02             LDA #2
0816- 8D A2 08          STA DRIVE         do drive 2 first

                DO.CATALOG
0819- 20 94 08          JSR CLEAR.BUFFER
081C- A9 11             LDA #$11
081E- 8D A4 08          STA TRACK
0821- 8D 00 09          STA MY.BUFFER+1   link pointer
0824- A0 1F             LDY #$1F
0826- 8C A5 08  .1      STY SECTOR
0829- 88                DEY
082A- D0 03             BNE .2
082C- 8C 00 09          STY MY.BUFFER+1   mark end of catalog
082F- 8C 01 09  .2      STY MY.BUFFER+2   link pointer
0832- 20 89 08          JSR CALL.RWTS
0835- AC A5 08          LDY SECTOR
0838- 88                DEY
0839- D0 EB             BNE .1            and go back for more
083B- 8C A5 08          STY SECTOR
```

```
                  1430 DO.VTOC
083E- 20 94 08    1440        JSR CLEAR.BUFFER
0841- A2 00       1450        LDX #0
0843- BC B1 08    1460 .1     LDY VTOC.INDEXES,X
0846- BD BC 08    1470        LDA VTOC.VALUES,X
0849- 99 FF 08    1480        STA MY.BUFFER,Y   set VTOC header info
084C- E8          1490        INX
084D- E0 0B       1500        CPX #ENTRY.COUNT
084F- 90 F2       1510        BCC .1
0851- AD A2 08    1520        LDA DRIVE         use drive # for volume
0854- 8D 05 09    1530        STA MY.BUFFER+6
0857- A9 FF       1540        LDA #$FF
0859- C8          1550        INY
085A- C8          1560 .2     INY               skip a track in bitmap
085B- C8          1570        INY
085C- C8          1580        INY
085D- C8          1590        INY
085E- 99 FF 08    1600 .3     STA MY.BUFFER,Y   mark free
0861- C8          1610        INY
0862- F0 06       1620        BEQ .4            leave if done
0864- C0 7C       1630        CPY #$7C          track $11?
0866- F0 F2       1640        BEQ .2            yes, skip it
0868- D0 F4       1650        BNE .3            no, go on
086A- 20 89 08    1660 .4     JSR CALL.RWTS
086D- CE A2 08    1670        DEC DRIVE         now go back and
0870- D0 A7       1680        BNE DO.CATALOG    do drive one
                  1690
                  1700 DO.BOOT.SECTOR
0872- EE A2 08    1710        INC DRIVE         that was drive one,
0875- 20 94 08    1720        JSR CLEAR.BUFFER  so write a boot sector
0878- 8D A4 08    1730        STA TRACK         A = 0
087B- 8D A5 08    1740        STA SECTOR
087E- A0 38       1750        LDY #BOOT.SIZE
0880- B9 C7 08    1760 .1     LDA BOOT.IMAGE,Y  install the image
0883- 99 FF 08    1770        STA MY.BUFFER,Y
0886- 88          1780        DEY
0887- 10 F7       1790        BPL .1            fall into CALL.RWTS
                  1800 *--------------------------------
                  1810 CALL.RWTS
0889- A9 08       1820        LDA /IOB
088B- A0 A0       1830        LDY #IOB
088D- 20 D9 03    1840        JSR RWTS
0890- B0 01       1850        BCS ERROR
0892- 60          1860        RTS
0893- 00          1870 ERROR  BRK
                  1880 *--------------------------------
                  1890 CLEAR.BUFFER
0894- A0 00       1900        LDY #0
0896- 98          1910        TYA
0897- 99 FF 08    1920 .1     STA MY.BUFFER,Y
089A- C8          1930        INY
089B- D0 FA       1940        BNE .1
089D- 60          1950        RTS
                  1960 *--------------------------------
089E- 01          1970 PC.PARMS .DA #1     one parm
089F- 01          1980          .DA #1     unit one
                  1990 *--------------------------------
08A0- 01          2000 IOB    .DA #1
08A1- 50          2010 SLOT   .DA #UNIDISK.SLOT*$10
08A2-             2020 DRIVE  .BS 1
08A3- 00          2030 VOL    .DA #0
08A4-             2040 TRACK  .BS 1
08A5-             2050 SECTOR .BS 1
08A6- FB B7       2060 DCT    .DA $B7FB
08A8- FF 08       2070 BUFFER .DA MY.BUFFER
08AA-             2080        .BS 1
08AB- 00          2090        .DA #0
08AC- 02          2100 COMAND .DA #2       write
08AD-             2110 RETURN .BS 1
08AE-             2120 P.VOL  .BS 1
08AF-             2130 P.SLOT .BS 1
08B0-             2140 P.DRIV .BS 1
                  2150 *--------------------------------
08B1- 00 01 02
08B4- 03 27 30
08B7- 31 34 35
08BA- 36 37       2160 VTOC.INDEXES .HS 00.01.02.03.27.30.31.34.35.36.37
0B-               2170 ENTRY.COUNT .EQ *-VTOC.INDEXES
```

```
08BC- 04 11 1F
08BF- 03 7A 11
08C2- 01 32 20
08C5- 00 01      2180 VTOC.VALUES  .HS 04.11.1F.03.7A.11.01.32.20.00.01
                 2190 *--------------------------------
                 2200 BOOT.IMAGE
                 2210             .PH $800
0800- 01         2220 BOOT        .HS 01
0801- 20 58 FC   2230             JSR HOME
0804- A0 00      2240             LDY #0
0806- B9 13 08   2250 .1          LDA MESSAGE,Y
0809- F0 06      2260             BEQ .2
080B- 20 ED FD   2270             JSR COUT          print message
080E- C8         2280             INY
080F- D0 F5      2290             BNE .1
0811- F0 FE      2300 .2          BEQ .2            and hang...
                 2310
                 2320 MESSAGE
0813- 8D 8D 8D   2330             .HS 8D8D8D
0816- D3 EF F2
0819- F2 F9 AC
081C- A0 E3 E1
081F- EE A7 F4
0822- A0 E2 EF
0825- EF F4 A0
0828- C4 CF D3
082B- A0 E8 E5
082E- F2 E5 A0
0831- F9 E5 F4
0834- AE         2340             .AS -/Sorry, can't boot DOS here yet./
0835- 8D 87 00   2350             .HS 8D8700
                 2360             .EP
38-              2370 BOOT.SIZE .EQ *-BOOT.IMAGE
                 2380 *--------------------------------
                 2390 MY.BUFFER
```

## Recovering & Repairing Lost Programs......Peter Bartlett, Jr.

Eldridge, Iowa

As a long-time user of the S-C Macro Assembler, I have learned a few tricks to save a lot of aggravation. Sometimes I mistakenly erase the source program I have in memory with the "NEW" or "LOAD" command. The program is not actually gone; instead, the pointer to the start of the program is changed.

At one time, I would adjust the source pointer by hand until my program was restored, but this was slow and painful. So like all good hackers I now have a little program to find the start of a program and adjust the pointer automatically.

My "Find.Start" program searches through memory for a source line numbered 1000 and resets the source pointer to that line. The search begins at HIMEM and proceeds down until it finds line 1000 or address $800.

The program itself is a simple search for the two-byte hex equivalent of 1000. On entry, the program starts the search at HIMEM and sets the "DONE.ONCE" flag so subsequent re-entries pick up the search where it last left off.

After the program stops, you can run it again to find the next lower source line numbered 1000. If several programs have been loaded into memory, you can run "Find.Start" several times to point to the start of each one.

The only way to start the search from HIMEM again is to re-load the program. It's not elegant, but does it really need to be?

In many instances, the next step is to re-construct the scrambled part of a program. This usually seems impossible, because the program's internal pointers will probably be scrambled and cause weird problems when editing.

Instead of fighting with the program (or hand-patching as I used to do), just use the handy "TEXT" command built into the assembler to create a text version of your program. Then enter the "AUTO" mode and "EXEC" the text version of your program back into memory. This will rectify all the internal pointers and leave you free to edit your program back into shape.

Perhaps that last paragraph is obvious, but I didn't think of it until recently. And we've had the "TEXT" command available for a long time!

```
                  1000 *SAVE FIND.START
                  1010 *--------------------------------
                  1020 *   SEARCH FROM HIMEM TO PP FOR LINE "1000"
                  1030 *   SET $CA,CB TO BEGINNING OF THAT LINE
                  1040 *--------------------------------
00-               1050 SRCP   .EQ $00,01
4C-               1060 HIMEM  .EQ $4C,4D
CA-               1070 PP     .EQ $CA,CB
                  1080 *--------------------------------
                  1090        .OR $300
                  1100 *--------------------------------
                  1110 DO
0300- A6 CA       1120        LDX PP       IF NOT FIRST TIME,
0302- A5 CB       1130        LDA PP+1          START WHERE WE LEFT OFF
0304- 2C 45 03    1140        BIT DONE.ONCE.FLAG
0307- 30 08       1150        BMI .1       ...NOT FIRST TIME
```

```
                        1160 *---HAS TO BE A FIRST TIME-------
0309- 38                1170        SEC           SET FLAG
030A- 6E 45 03          1180        ROR DONE.ONCE.FLAG
030D- A6 4C             1190        LDX HIMEM     START AT TOP OF SOURCE AREA
030F- A5 4D             1200        LDA HIMEM+1
                        1210 *---STORE STARTING POINTER-------
0311- 86 00             1220 .1     STX SRCP
0313- 85 01             1230        STA SRCP+1
0315- 20 3C 03          1240        JSR DEC.SRCP
                        1250 *---SEARCH FOR "1000"------------
0318- 20 3C 03          1260 .2     JSR DEC.SRCP
031B- A5 01             1270        LDA SRCP+1
031D- C9 08             1280        CMP /$0800    DON'T SEARCH BEYOND $800
031F- 90 1A             1290        BCC .3        ...END OF SEARCH
0321- A0 00             1300        LDY #0
0323- B1 00             1310        LDA (SRCP),Y
0325- C9 E8             1320        CMP #1000     COMPARE LO-BYTE
0327- D0 EF             1330        BNE .2        ...NO, KEEP SCANNING
0329- C8                1340        INY           ...MATCH, CHECK HI-BYTE
032A- B1 00             1350        LDA (SRCP),Y
032C- C9 03             1360        CMP /1000
032E- D0 E8             1370        BNE .2        ...NO, KEEP SCANNING
                        1380 *---FOUND IT, POINT PP TO IT-----
0330- 20 3C 03          1390        JSR DEC.SRCP  BACK UP OVER BYTE COUNT
0333- A5 00             1400        LDA SRCP
0335- 85 CA             1410        STA PP
0337- A5 01             1420        LDA SRCP+1
0339- 85 CB             1430        STA PP+1
033B- 60                1440 .3     RTS
                        1450 *--------------------------------
                        1460 DEC.SRCP
033C- A5 00             1470        LDA SRCP
033E- D0 02             1480        BNE .1
0340- C6 01             1490        DEC SRCP+1
0342- C6 00             1500 .1     DEC SRCP
0344- 60                1510        RTS
                        1520 *--------------------------------
0345- 00                1530 DONE.ONCE.FLAG .HS 00
                        1540 *--------------------------------
```

More and Better Division by Seven.........Bob Sander-Cederlof

I can think of at least three good reasons we need a good subroutine for dividing by seven. We need it in computations involving the day of week. We need it in hi-res graphics programs to calculate the byte and bit for a particular pixel between 0 and 279 for normal hi-res, or between 0 and 559 for double hi-res. Lastly, the new protocol converter interface used in connection with the Unidisk 3.5 works with packets of up to 767 bytes which are made up of a number of 7-byte groups.

In looking through the assembly listing of the new //c ROMs, which come with the Unidisk 3.5 update, I noticed a divide-by-seven subroutine at $CB45-CBAF. The code divides the buffer size, which can be up to $2FF, by seven, and saves both the quotient and the remainder. The code looks too large and too slow and too complicated ... in other words, it looks like a challenging assignment. My transposition of the //c code follows, and as I count cycles it takes from 133 to 268 cycles depending on the value of the dividend. The code and tables take 71 bytes in the //c ROM.

While I was musing on the possibilities, Michael Hackney called me from Troy, New York. He wondered if we were interested in publishing his fast 65802 routine for dividing by seven. Michael uses his in a speedy double hi-res program. He divides values up to 559 ($22F) by seven, keeping both the quotient and remainder, in 66 cycles. Michael's subroutine itself is short (37 bytes), but he uses a 140-byte table to achieve the speed. Adding another 84 bytes to the tables extends the range to handle dividends up to 895 ($37F).

(In all the times and lengths given here, I am not counting the JSR-RTS cycles nor the RTS byte. I assume the code is critical enough that it would be placed in-line in actual use, rather than made into a JSR-called subroutine. I am also not counting any overhead I added to switch from 65802 mode to 6502 and back, as this was only added due to my test program being in 65802 mode. All of the subroutines use page zero for variable and temporary storage. They would be longer and slightly slower if the variables and temporaries were not in page zero.)

Yesterday I spent the whole day dividing by seven. I came up with two new subroutines: one for the 65802, and one for a normal 6502. They are both small and fast. First I tackled the 65802 version, and based in on multiplying by 1/7 as a binary fraction. This one came out 39 bytes long, executing in 64 cycles. This one used a fudge factor; the largest dividend it can handle is 594 ($252). By using alternate code to extend the precision, numbers up to 895 ($37F) can be handled. This one takes the same number of bytes, but 9 cycles longer.

Finally, I wrote a normal 6502 version. Strangely enough, it came out only 60 bytes long and only 76 cycles! Makes me wonder if I couldn't do better in the 65802, given another day or two. The 6502 version handles dividends up to 1023 ($3FF). It would be two bytes shorter if the range was restricted to $2FF.

Here is a table summarizing the size, timing, and dividend range for the various subroutines:

| | bytes | cycles | dividend |
|---|---|---|---|
| //c ROM | 71 | 133-268 | 0-$2FF |
| Hackney 65802 | 177 | 66 | 0-$22F |
| RBSC 65802-1 | 39 | 64 | 0-$252 |
| RBSC 65802-2 | 39 | 73 | 0-$37F |
| RBSC 6502 | 60 | 76 | 0-$3FF |

The listing which follows includes all five versions, plus a testing program. The testing program runs through the entire range from $3FF down to 0. After doing the division by the selected method, a check subroutine tests for a valid remainder (a number less than 7); it further tests that the quotient*7 +remainder = the original dividend. If not, the dividend, quotient, and remainder are all printed in hexadecimal. If they are correct, the next dividend is tried. A keyboard pausing subroutine allows you to stop the display momentarily and/or abort the test run.

Lines 1020-1060 control some conditional assembly which select which division method to use. By change the value of VERSION in line 1020 I can assemble any one of the four routines. I used the "CON" listing option in line 1180 (which is not itself listed: it is "1180    .LIST CON") so that you can see what the un-assembled lines of code are. Other conditional code at lines 1720-1860 and 4010-4050 selects options mentioned above.

Lines 1200-1540 control each test run. I wrote this program using 65802 instructions, although it would not be difficult to re-write it for a plain 6502. Lines 1210-1220 enter the 65802 Native Mode. and lines 1520-1530 leave it. It is VERY IMPORTANT to be sure you do not exit a program and return to normal Apple software while still in the Native Mode. The most fantastic things can happen if you forget!

Lines 1580-1950 are my 65802 version. This entire subroutine is executed in the 65802 native mode, with the M-bit set so the A-register operations are 16-bits. The value 1/7 in binary is .001001001001001...forever. Multiplying by than number should give the same answer as dividing by seven. It also has the surprising side effect that the three bits after the "quotient" portion of the product will be equal to the "remainder". The values of the fractions from 0/7 to 6/7 are just nice that way:

| fraction | repeating decimal | same value in hex | the first three bits |
|---|---|---|---|
| 0/7 | .000000 | .000 | 000 |
| 1/7 | .142857.. | .249.. | 001 |
| 2/7 | .285714.. | .492.. | 010 |
| 3/7 | .428571.. | .6DB.. | 011 |
| 4/7 | .571428.. | .924.. | 100 |
| 5/7 | .714285.. | .B6D.. | 101 |
| 6/7 | .857142.. | .DB6.. | 110 |

Wow! Isn't that neat? More justification for the numerologists who claim that seven is the "perfect" number.

Now it remains to find the most efficient way to multiply by that fraction. The method I came up with first forms the product for .01000001 (lines 1600-1670). Then I divide that result by 8, which is the product for .0001000001 (lines 1680-1700). Adding the two products in line 1710 gives me the product for .0100100101 (approximately 2/7). Dividing that by two gives me an approximation for the division by seven. The code that follows in lines 1720-1800 is not assembled, because of the ".DO 0" line. What it does is extend the multiplication to include one more partial product. The shortest way I could think of to get that little number is demonstrated in the code you see. The extra precision makes my subroutine work for dividends up to $37F. It fails above that value because of overflow during the multiplication. If I leave out the extra precision, the subroutine gets the wrong answers for some numbers at each end of the range. By adding a "fudge factor" (a trick learned in college laboratory assignments to force experimental results to fit the laws of science), I can make all the dividends up to $252 work. The fudge factor adds $000A for values in the A-register of $8800 or more, and only $0008 for values below $8800.

Line 1870 is the division by two mentioned above. Lines 1880-1940 shift the first three bits of the remainder over to the correct position in the lower byte of the A-register. As I was writing the previous sentence, it suddenly struck me that the second set of three bits might be the same as the first set, if my multiplications happened to be precise enough. I went back to the assembler, changed line 1720 to ".DO 1" so the more precise version would assemble, and then replaced lines 1910-1930 with "1910    AND #7". Guess what! It worked! One byte shorter and four cycles faster! That makes it 38 bytes long, and only 69 cycles.

Next is my 6502 version, lines 1970-2370. The first four lines simply save the current state of the M and X bits, and the mode, and switch to 6502 emulation mode. They are matched by lines 2340-2360, which restore the mode and state. These will work regardless of what mode and state the machine was in when the subroutine was called. Since the subroutine would normally only be used in a 6502, you would leave out lines 1980-2010 and 2340-2360. I did not count them when timing the code. Back in December of 1984 I wrote in these pages of a nifty way to divide a one-byte value by seven. I used that method here, for dividing the low-order byte of the dividend. I then computed the remainder by multiplying the quotient by 7 and subtracting it from the dividend. Saving that quotient and remainder, I used a table lookup to determine the quotient and remainder of the high-order byte of the number. Since it could only have the values 0-3, the tables are very short. Then I add the two remainders together, modulo 7; and the two quotients, remembering the carry from the remainder if any.

Lines 2030-2170 are essentially the same as published in that December issue of AAL, except for the addition of lines 2130,

2140, and 2160. With those two lines I am saving a few steps in the multiplication by seven that I must do. Lines 2190-2200 finish the multiplication by seven, by adding the *2 and *4 values saved above. Lines 2210-2200 form the complement of the value. so I can subtract by adding. Normally a complement is formed by:

```
        EOR #$FF
        CLC
        ADC #1
```

I do the same with two less bytes and cycles here by preceding the addition at line 2230 with SEC rather than the usual CLC. I saved a byte and two cycles by storing one less than the actual remainder in the table of remainders at line 2400.

Lines 2420-2640 are called to print out the results when they don't meet expectations. Notice lines 2430-2460 and 2610-2630, which make sure I am in the correct state and mode. The monitor routines will not work correctly in 16-bit state, and may not work correctly in 65802 Native mode.

Lines 2660-2920 check the results. The subroutine returns with carry clear if the quotient and remainder are correct, or carry set if they are not. I check both by multiplying the quotient by seven and adding the remainder to see if the result equals the dividend, and I also make sure the remainder is less than seven. It is possible to get an answer with the quotient one less than it should be and a remainder of 7, so I had to test the remainder.

The PAUSE routine checks to see if any key has been typed. If so, and if it is not a <RETURN>, it waits until another key is typed. Note that I had to set 8-bit mode, to prevent the softswitch at $C011 from being switched. This also makes the CMP work properly. Otherwise the LDA $C000 would get two copies of the same character in the two halves of the A-register.

Lines 3060-3540 are essentially the code from the new //c ROMs. I re-arranged it a little, to make a stand-alone routine within my test-bed, and I changed labels and variable names. Apple uses two sets of tables. One gives quotients and remainders for 0, $100, and $200 (the high byte of the dividend). The other gives quotients and remainders for 0, $08, $10, $20, $40, and $80. A loop runs 5 times to add in the quotients and remainders for bits 3-7 of the dividend, and then fakes one more trip to add in the value of bits 0-2. Not efficient!

Michael Hackney's code is in lines 3560-4080. I'll quote from his letter.

"Apple hi-res graphics characteristically involve various calculations to determine the exact display address from a given X,Y pair. Typically, the vertical position (Y) base address is found by table look-up. The horizontal, or X, position is determined by dividing by 7 (since there are seven pixel bits per byte in the hi-res screen). The integer portion

of the division is the byte offset from the base address, and the remainder is the position in the byte. Brute calculation (which is slow for graphics routines) or table lookup (which takes a lot of space) is used to do the division. Table lookup is usually used in good graphics programs. Hi-res graphics require two 280-byte tables, one for quotient and one for remainder. Double hi-res requires tables twice as big. My interest in 65802/816 double-he-res graphics drivers has prompted me to find a serviceable divide-by-seven which is quick and doesn't require more than one page of memory.

"The 65802/816 16-bit operations are ideally suited for this task. Larger numbers can be easily manipulated and table lookup can retrieve 2 bytes of data at once. My routine uses both of these techniques to perform its duty. It divides the original number by eight before doing any table lookup (this keeps the table smaller). The it mulitplies both the quotient and remainder retrieved from the table by 8. The resulting remainder is added to the original lower three bits (the ones shifted out when I divided by 8), and I look into the table again. The first quotient is added to the second quotient, and it is finished. The table only takes 140 bytes, storing quotients and remainders for numbers up to 69. Everything fits in a page with room to spare.

"As an extra bonus, I included a small routine which generates the table in situ. The area occupied by the table generator can be used for data storage once the table is built. It takes longer to load a table from disk than it does to compute one, and the generator dissappears after use, so this is the best way to do it."

In order to get the greatest speed, Michael's table should all reside entirely in the same page of memory. That is why I included line 4100, which justifies the table to the beginning of the next page.

So here you have four great answers to the challenge. Now it's your turn!

```
                    1000 *SAVE BETTER DIV 7+
                    1010 *--------------------------------
01-                 1020 VERSION     .EQ 1
01-                 1030 RBSC65802   .EQ 1
02-                 1040 HACKNEY     .EQ 2
03-                 1050 TWO.C       .EQ 3
04-                 1060 RBSC6502    .EQ 4
                    1070 *--------------------------------
00-                 1080 DIVIDEND    .EQ 0,1
02-                 1090 QUO.REM     .EQ 2,3
04-                 1100 T1          .EQ 4,5
06-                 1110 T2          .EQ 6,7
                    1120 *--------------------------------
FD8E-               1130 CROUT  .EQ $FD8E
FDDA-               1140 PRBYTE .EQ $FDDA
FDED-               1150 COUT   .EQ $FDED
                    1160 *--------------------------------
                         1170        .OP 65802

                         1190 *--------------------------------
                         1200 TEST
000800- 18               1210        CLC          ENTER NATIVE MODE
000801- FB               1220        XCE
                         1230   .DO VERSION=HACKNEY
                         1240        JSR BUILD.HACKNEY.TABLE
                         1250   .FIN
```

```
000802- C2 20                 REP #$20     16-BIT A-REGISTER
000804- A9 FF 03              LDA ##$3FF   LARGEST VALUE TO TEST
000807- 85 00                 STA DIVIDEND
000809- A5 00           .1    LDA DIVIDEND
                         .DO VERSION=RBSC65802
00080B- 20 26 08              JSR DIVIDE.BY.SEVEN.65802
00080E- 85 02                 STA QUO.REM  QUO IN 15...8, REM IN 7...0
                         .FIN
                         .DO VERSION=HACKNEY
                              JSR HACKNEY.DIV7
                              STA QUO.REM  QUO IN 15...8, REM IN 7...0
                         .FIN
                         .DO VERSION=RBSC6502
                              JSR DIVIDE.BY.SEVEN.6502
                         .FIN
                         .DO VERSION=TWO.C
                              JSR DIV7.TWOC
                         .FIN
000810- 20 BD 08              JSR CHECK    TEST RESULT BY MULTIPLYING
000813- 90 03                 BCC .2       ...CORRECT ANSWER
000815- 20 92 08              JSR PRINT    ...INCORRECT DIVISION
000818- 20 E8 08        .2    JSR PAUSE    CHECK FOR KEYPRESS
00081B- F0 06                 BEQ .3       <RET>, ABORT
00081D- C2 20                 REP #$20     16-BIT A-REGISTER
00081F- C6 00                 DEC DIVIDEND
000821- 10 E6                 BPL .1       ...NEXT ONE
000823- 38              .3    SEC          RETURN TO EMULATION MODE
000824- FB                    XCE
000825- 60                    RTS
                        *--------------------------------
                        *   QUO = VAL * .001001001001001
                        *--------------------------------
                        DIVIDE.BY.SEVEN.65802
000826- 85 04                 STA T1       SAVE ORIGINAL VALUE
000828- 0A                    ASL          MULTIPLY BY 64
000829- 0A                    ASL
00082A- 0A                    ASL
00082B- 0A                    ASL
00082C- 0A                    ASL
00082D- 0A                    ASL
00082E- 65 04                 ADC T1       ADD, EQUIV. TO * .01000001
000830- 85 04                 STA T1       SAVE RESULT
000832- 4A                    LSR          DIVIDE BY 8, WHICH IS
000833- 4A                    LSR               EQUIV. TO * .00001000001
000834- 4A                    LSR
000835- 65 04                 ADC T1       EQUIV TO * .01001001001
                         .DO 0
                              STA T1       EXTENDED PRECISION METHOD
                              XBA          GET EQUIV. TO * .00000000000001
                              AND ##$00FF
                              LSR
                              LSR
                              LSR
                              LSR
                              ADC T1       EQUIV. TO * .01001001001001
                         .ELSE
000837- C9 00 88              CMP ##$8800  FUDGE FACTOR METHOD
00083A- 69 08 00              ADC ##$0008  ADD $0008 TO ALL VALUES,
00083D- C9 00 88              CMP ##$8800       AND $0002 MORE TO BIG ONES
000840- 69 00 00              ADC ##$0000
                         .FIN
000843- 4A                    LSR          DIVIDE BY 2, RESULT IS QUOTIENT
000844- E2 20                 SEP #$20          IN HI BYTE, REM IN NEXT 3 BITS
000846- 4A                    LSR          ISOLATE REMAINDER IN LO BYTE
000847- 4A                    LSR
000848- 4A                    LSR
000849- 4A                    LSR
00084A- 4A                    LSR
00084B- C2 20                 REP #$20
00084D- 60                    RTS
                        *--------------------------------
                        DIVIDE.BY.SEVEN.6502
00084E- 08                    PHP          SAVE M&X BITS
00084F- 38                    SEC          SWITCH TO EMULATION MODE
000850- FB                    XCE
000851- 08                    PHP
                        *--------------------------------
```

```
000852- A5 00                LDA DIVIDEND
000854- 4A                   LSR
000855- 4A                   LSR
000856- 4A                   LSR
000857- 65 00                ADC DIVIDEND
000859- 6A                   ROR
00085A- 4A                   LSR
00085B- 4A                   LSR
00085C- 65 00                ADC DIVIDEND
00085E- 6A                   ROR
00085F- 29 FC                AND #$FC
000861- 85 04                STA T1
000863- 4A                   LSR
000864- 85 06                STA T2
000866- 4A                   LSR
000867- 85 03                STA QUO.REM+1      QUO = LO-BYTE/7
000869- 65 04                ADC T1
00086B- 65 06                ADC T2             QUO*7
00086D- 49 FF                EOR #$FF           -QUO*7
00086F- 38                   SEC
000870- 65 00                ADC DIVIDEND       REM
000872- A6 01                LDX DIVIDEND+1     0,1, OR 2
000874- 7D 8E 08             ADC RTBL,X
000877- C9 07                CMP #7
000879- 90 02                BCC .1
00087B- E9 07                SBC #7
00087D- 85 02           .1   STA QUO.REM        FINAL REMAINDER
00087F- BD 8A 08             LDA QTBL,X
000882- 65 03                ADC QUO.REM+1
000884- 85 03                STA QUO.REM+1      FINAL QUOTIENT
                        *--------------------------------
000886- 28                   PLP           SWITCH TO ORIGINAL MODE
000887- FB                   XCE
000888- 28                   PLP           X&M BITS
000889- 60                   RTS
                        *--------------------------------
00088A- 00 24 49 6D     QTBL .DA #0,#36,#73,#109
00088E- FF 03 00 04     RTBL .DA #-1,#3,#0,#4
                        *--------------------------------
                        PRINT
000892- 08                   PHP           SAVE M&X BITS
000893- 38                   SEC           SWITCH TO EMULATION MODE
000894- FB                   XCE
000895- 08                   PHP           SAVE ORIGINAL MODE (C-BIT)
000896- A5 01                LDA DIVIDEND+1
000898- 09 B0                ORA #"0"      PRINT DIVIDEND IN HEX
00089A- 20 ED FD             JSR COUT
00089D- A5 00                LDA DIVIDEND
00089F- 20 DA FD             JSR PRBYTE
0008A2- A9 A0                LDA #" "      PRINT QUOTIENT IN HEX
0008A4- 20 ED FD             JSR COUT
0008A7- A5 03                LDA QUO.REM+1
0008A9- 20 DA FD             JSR PRBYTE
0008AC- A9 A0                LDA #" "      PRINT REMAINDER IN HEX
0008AE- 20 ED FD             JSR COUT
0008B1- A5 02                LDA QUO.REM
0008B3- 20 DA FD             JSR PRBYTE
0008B6- 20 8E FD             JSR CROUT     <RETURN>
0008B9- 28                   PLP           RESTORE NATIVE/EMULATION BIT
0008BA- FB                   XCE
0008BB- 28                   PLP           RESTORE M&X BITS
0008BC- 60                   RTS
                        *--------------------------------
                        CHECK
0008BD- A5 02                LDA QUO.REM
0008BF- 29 00 FF             AND ##$FF00   ISOLATE QUOTIENT
0008C2- 4A                   LSR           DIVIDE BY 64 FOR NOW
0008C3- 4A                   LSR
0008C4- 4A                   LSR
0008C5- 4A                   LSR
0008C6- 4A                   LSR
0008C7- 4A                   LSR
0008C8- 85 04                STA T1
0008CA- 4A                   LSR           MULTIPLY BY SEVEN
0008CB- 85 06                STA T2
0008CD- 4A                   LSR
0008CE- 65 04                ADC T1
0008D0- 65 06                ADC T2
```

```
0008D2- 85 04       2810        STA T1        QUO * 7
0008D4- A5 02       2820        LDA QUO.REM   CHECK FOR VALID REMAINDER
0008D6- 29 FF 00    2830        AND ##$00FF   0...7
0008D9- C9 07 00    2840        CMP ##7
0008DC- B0 08       2850        BCS .1        ...INVALID REMAINDER
0008DE- 65 04       2860        ADC T1        ADD QUO*7
0008E0- C5 00       2870        CMP DIVIDEND  ...BETTER BE SAME!
0008E2- D0 02       2880        BNE .1        ...NOT, INVALID QUO & REM
0008E4- 18          2890        CLC           SIGNAL VALID ANSWERS
0008E5- 60          2900        RTS
0008E6- 38          2910 .1     SEC           SIGNAL INVALID ANSWERS
0008E7- 60          2920        RTS
                    2930 *--------------------------------
                    2940 PAUSE
0008E8- E2 20       2950        SEP #$20      8-BIT A-REGISTER
0008EA- AD 00 C0    2960        LDA $C000     CHECK KEYBOARD
0008ED- 10 0F       2970        BPL .2        NOTHING TYPED
0008EF- 8D 10 C0    2980        STA $C010     CLEAR STROBE
0008F2- C9 8D       2990        CMP #$8D      <RETURN>?
0008F4- F0 08       3000        BEQ .2        <RET>, SO DON'T PAUSE
0008F6- AD 00 C0    3010 .1     LDA $C000     SOME OTHER KEY, SO PAUSE
0008F9- 10 FB       3020        BPL .1        ...TILL ANOTHER KEY TYPED
0008FB- 8D 10 C0    3030        STA $C010     CLEAR STROBE
0008FE- C9 8D       3040 .2     CMP #$8D      .EQ. IF <RET>
000900- 60          3050        RTS           ...ELSE .NE.
                    3060 *--------------------------------
                    3070 *   DIVIDE BY 7 FROM NEW //C ROMS (AT $CB4F-CBB0)
                    3080 *      USED TO GET NUMBER OF 7-BYTES PACKETS
                    3090 *      IN A BUFFER, FOR THE PROTOCOL CONVERTER
                    3100 *--------------------------------
                    3110 DIV7.TWOC
000901- 08          3120        PHP           SAVE X&M BITS
000902- 38          3130        SEC           ENTER EMULATION MODE
000903- FB          3140        XCE
000904- 08          3150        PHP           SAVE PREVIOUS MODE
                    3160 *---ALGORITHM FROM //C-----------
000905- A6 01       3170        LDX DIVIDEND+1     HI BYTE (0, 1, OR 2)
000907- BD 40 09    3180        LDA PDIV7TAB,X   0, $100, OR $200 DIVIDED BY 7
00090A- 85 03       3190        STA QUO.REM+1   QUOTIENT SO FAR
00090C- BD 43 09    3200        LDA PMOD7TAB,X   0, $100, OR $200 MOD 7
00090F- 85 02       3210        STA QUO.REM      REMAINDER SO FAR
                    3220 *---PROCESS NEXT 5 BITS----------
000911- A2 05       3230        LDX #5
000913- A5 00       3240        LDA DIVIDEND       LOW BYTE
000915- 85 04       3250        STA T1             WORKING COPY
000917- 29 07       3260        AND #7             LOW 3 BITS
000919- A8          3270        TAY           SAVE FOR LATER USE
00091A- 06 04       3280 .1     ASL T1        GET NEXT BIT FROM DIVIDEND IN CARRY
00091C- 90 15       3290        BCC .4        IF CLEAR, NO EFFECT ON QUO,MOD
00091E- BD 46 09    3300        LDA MOD7TAB,X      GET MOD7 FOR 2^N
000921- 18          3310 .2     CLC           UPDATE MOD VALUE
000922- 65 02       3320        ADC QUO.REM
000924- C9 07       3330        CMP #7        OVERFLOW?
000926- 90 02       3340        BCC .3        ...NO
000928- E9 07       3350        SBC #7        ...YES, CORRECT
00092A- 85 02       3360 .3     STA QUO.REM   REMAINDER SO FAR
00092C- BD 4C 09    3370        LDA DIV7TAB,X      GET QUOTIENT FOR 2^N
00092F- 65 03       3380        ADC QUO.REM+1
000931- 85 03       3390        STA QUO.REM+1      QUOTIENT SO FAR
000933- CA          3400 .4     DEX                ONE LESS BIT TO DEAL WITH
000934- 30 06       3410        BMI .5             ...FINISHED
000936- D0 E2       3420        BNE .1             ...FIVE TIMES
000938- 98          3430        TYA                GET BACK FIRST 3 BITS
000939- 4C 21 09    3440        JMP .2             ADD IN REMAINDER
                    3450 *---RETURN TO CALLER-------------
00093C- 28          3460 .5     PLP           ORIGINAL MODE
00093D- FB          3470        XCE
00093E- 28          3480        PLP           RESTORE X&M BITS
00093F- 60          3490        RTS
                    3500 *--------------------------------
000940- 00 24 49    3510 PDIV7TAB .DA #0,#36,#73
000943- 00 04 01    3520 PMOD7TAB .DA #0,#4,#1
000946- 00 01 02 04
00094A- 01 02       3530 MOD7TAB .DA #0,#1,#2,#4,#1,#2
00094C- 00 01 02 04
000950- 09 12       3540 DIV7TAB .DA #0,#1,#2,#4,#9,#18
                    3550 *--------------------------------
```

```
                        3560 HACKNEY.DIV7
000952- 85 04           3570        STA T1          SAVE VALUE
000954- 29 07 00        3580        AND ##$0007     SAVE LOWER 3 BITS (MOD 8)
000957- 85 06           3590        STA T2
000959- A5 04           3600        LDA T1          DIVIDE BY 8
00095B- 4A              3610        LSR
00095C- 4A              3620        LSR
00095D- 4A              3630        LSR
00095E- 0A              3640        ASL             DOUBLE FOR TABLE INDEX
00095F- AA              3650        TAX             GET QUO & REM FROM TABLE
000960- BD 00 0A        3660        LDA TABLE,X
000963- 0A              3670        ASL             MULTIPLY BOTH BY 8
000964- 0A              3680        ASL
000965- 0A              3690        ASL
000966- 65 06           3700        ADC T2          ADD LOWER BITS BACK
000968- AA              3710        TAX             SAVE RESULT
000969- 29 00 FF        3720        AND ##$FF00     KEEP QUOTIENT
00096C- 85 04           3730        STA T1
00096E- 8A              3740        TXA             GET REMAINDER
00096F- 0A              3750        ASL             DOUBLE FOR INDEX
000970- AA              3760        TAX
000971- BD 00 0A        3770        LDA TABLE,X     GET QUO & REM FROM TABLE
000974- 18              3780        CLC             ADD PREVIOUS QUOTIENT
000975- 65 04           3790        ADC T1
000977- 60              3800        RTS
                        3810 *--------------------------------
                        3820 BUILD.HACKNEY.TABLE
000978- 08              3830        PHP             SAVE M&X BITS
000979- C2 20           3840        REP #$20        LONG A-REG
00097B- A9 00 0A        3850        LDA ##TABLE
00097E- 85 04           3860        STA T1
000980- E2 30           3870        SEP #$30        ALL REGS SHORT
000982- A2 00           3880        LDX #0          X = REMAINDER
000984- 9B              3890        TXY             Y = QUOTIENT
000985- 8A              3900 .1     TXA             STORE CURRENT REMAINDER
000986- 92 04           3910        STA (T1)
000988- E6 04           3920        INC T1
00098A- 98              3930        TYA             STORE CURRENT QUOTIENT
00098B- 92 04           3940        STA (T1)
00098D- E6 04           3950        INC T1
00098F- E8              3960        INX             NEXT REMAINDER
000990- E0 07           3970        CPX #7
000992- 90 F1           3980        BCC .1          ...NO CHANGE TO QUOTIENT
000994- A2 00           3990        LDX #0          NEXT QUOTIENT
000996- C8              4000        INY
                        4010   .DO 1
000997- C0 0A           4020        CPY #10         STOP AFTER QUO=9, REM=6
                        4030   .ELSE
                        4040        CPY #16         STOP AFTER QUO=15, REM=6
                        4050   .FIN
000999- 90 EA           4060        BCC .1          ...NOT YET
00099B- 28              4070        PLP             RESTORE M&X BITS
00099C- 60              4080        RTS
                        4090 *--------------------------------
00099D-                 4100        .BS *+255/256*256-*
0A00-                   4110 TABLE  .EQ *
                        4120 *--------------------------------
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $45 per year in the US, Canada and Mexico, and $87 to other countries.